MiniMoonBit 2024 程序设计语言规范、文法及说明

by MiniMoonBit Authors

§1. 语法定义

MiniMoonBit 的语法定义如下，以 ANTLR 语法编写。你也可以在 我们的仓库里 找到同样的内容。

```
grammar MiniMoonBit;

prog: top_level* EOF;

// Top-level
//
// Top level declarations should start at the beginning of the line, i.e.
// token.column == 0. Since this is non-context-free, it is not included in this
// backend-agnostic ANTLR grammar.
top_level: top_let_decl | toplevel_fn_decl;
top_let_decl:
  'let' IDENTIFIER ':' type '=' expr ';';
toplevel_fn_decl: (main_fn_decl | top_fn_decl) ';';

// Function declarations
//
// `fn main` and `fn init` does not accept parameters and return type
main_fn_decl: 'fn' ('main' | 'init') fn_body;

top_fn_decl:
  'fn' IDENTIFIER '(' param_list? ')' '->' type fn_body;
param_list: param (',' param)*;
param: IDENTIFIER type_annotation;
fn_body: '{' stmt '}';

nontop_fn_decl:
  'fn' IDENTIFIER '(' nontop_param_list? ')' (
    '->' type
  )? fn_body;
nontop_param_list:
  nontop_param (',' nontop_param)*;
nontop_param: IDENTIFIER type_annotation?;

// Statements
stmt:
  let_tuple_stmt
  | let_stmt
  | fn_decl_stmt
  | assign_stmt
  | expr_stmt;

let_tuple_stmt:
  'let' '(' IDENTIFIER (',' IDENTIFIER)* ')' type_annotation? '=' expr ';'
    stmt;
let_stmt:
  'let' IDENTIFIER type_annotation? '=' expr ';' stmt;
```

```
type_annotation: COLON type;

fn_decl_stmt: top_fn_decl ';' stmt;

assign_stmt:
  get_expr '=' expr ';' stmt; // x[y] = z;
expr_stmt: expr;

// Expressions, in order of precedence.
expr:
  add_sub_level_expr
  | add_sub_level_expr '==' expr
  | add_sub_level_expr '<=' expr;

add_sub_level_expr:
  mul_div_level_expr
  | mul_div_level_expr '+' add_sub_level_expr
  | mul_div_level_expr '-' add_sub_level_expr;

mul_div_level_expr:
  if_level_expr
  | if_level_expr '*' mul_div_level_expr
  | if_level_expr '/' mul_div_level_expr;

if_level_expr: get_or_apply_level_expr | if_expr;
if_expr: 'if' expr block_expr ('else' block_expr)?;

get_or_apply_level_expr:
  get_expr
  | apply_expr
  | value_expr;
get_expr: value_expr '[' expr ']'; // x[y]
apply_expr: empty_apply_expr | nonempty_apply_expr;
empty_apply_expr: value_expr '(' ')'; // f()
nonempty_apply_expr:
  value_expr '(' expr (',' expr)* ')'; // f(x, y)

// Value expressions
value_expr:
  | unit_expr
  | tuple_expr
  | bool_expr
  | identifier_expr
  | block_expr
  | neg_expr
  | floating_point_expr
  | int_expr
  | not_expr
  | array_make_expr;
unit_expr: '(' ')'; // ()
tuple_expr: '(' expr (',' expr)* ')'; // (x, y)
block_expr: '{' stmt '}'; // { blah; blah; }
bool_expr: 'true' | 'false';
neg_expr: '-' value_expr;
floating_point_expr: NUMBER '.' NUMBER?; // 1.0 | 1.
int_expr: NUMBER; // 1
not_expr: 'not' '(' expr ')'; // not(x)
array_make_expr:
  'Array' ':' ':' 'make' '(' expr ',' expr ')'; // Array::make(x, y)
```

```
identifier_expr: IDENTIFIER;

// Types
type:
  'Unit'
  | 'Bool'
  | 'Int'
  | 'Double'
  | array_type
  | tuple_type
  | function_type;
array_type: 'Array' '[' type ']';
tuple_type: '(' type (',' type)* ')'; // (Int, Bool)
function_type:
  '(' type (',' type)* ')' '->' type; // (Int, Bool) → Int

// Tokens

TRUE: 'true';
FALSE: 'false';
UNIT: 'Unit';
BOOL: 'Bool';
INT: 'Int';
DOUBLE: 'Double';
ARRAY: 'Array';
NOT: 'not';
IF: 'if';
ELSE: 'else';
FN: 'fn';
LET: 'let';
NUMBER: [0-9]+;
IDENTIFIER: [a-zA-Z_][a-zA-Z0-9_]*;
DOT: '.';
ADD: '+';
SUB: '-';
MUL: '*';
DIV: '/';
ASSIGN: '=';
EQ: '==';
LE: '<=';
LPAREN: '(';
RPAREN: ')';
LBRACKET: '[';
RBRACKET: ']';
LCURLYBRACKET: '{';
RCURLYBRACKET: '}';
ARROW: '->';
COLON: ':';
SEMICOLON: ';';
COMMA: ',';
WS: [ \t\r\n]+ → skip;
```

§2. 预定义的函数

运行环境中需要预先定义以下辅助函数：

```
// 输入输出函数
/// 读取一个整数，如果失败返回 INT_MIN
fn read_int() -> Int;
/// 打印一个整数，不带换行
fn print_int(i: Int) -> Unit;
/// 读取一个字节，如果失败返回 -1
fn read_char() -> Int;
/// 打印一个字节
fn print_char(c: Int) -> Unit;
/// 打印一个换行
fn print_newline() -> Unit;

// 数学函数
/// 整数和浮点数的互相转换
fn int_of_float(f: Double) -> Int;
fn float_of_int(i: Int) -> Double;
fn truncate(f: Double) -> Int;  // 与 int_of_float 相同
/// 浮点数运算
fn floor(f: Double) -> Double;
fn abs_float(f: Double) -> Double;
fn sqrt(f: Double) -> Double;
fn sin(f: Double) -> Double;
fn cos(f: Double) -> Double;
fn atan(f: Double) -> Double;
```

我们会以标准 RISC-V C 调用约定在提供这些函数的实现，实现的名称为实际函数名称前加入 `minimbt_`，如 `minimbt_print_int`。 在实现时，你可以认为所有不在作用域中的函数名称都是外部函数，并在函数名前加入 `minimbt_` 转换为外部调用。

此外，为了实现闭包、数组、元组等特性，我们还提供了以下内存分配函数：

```
/// 内存分配函数
void* minimbt_malloc(int32_t size);
/// 分配对应大小的内存，并初始化所有元素为给定的值
int32_t* minimbt_create_array(int32_t n_elements, int32_t init);
double* minimbt_create_float_array(int32_t n_elements, double init);
void** minimbt_create_ptr_array(int32_t n_elements, void* init);
```

你将不需要释放内存。

§3. 语义

§3.1. Typing

MiniMoonBit 的类型规则如下：

§3.1.1. Value expressions

$$\frac{}{\vdash \texttt{()} : \texttt{Unit}} \quad (\text{T-Unit})$$

$$\frac{}{\vdash \texttt{true} : \texttt{Bool}} \quad (\text{T-Bool-true})$$

$$\frac{}{\vdash \texttt{false} : \texttt{Bool}} \quad (\text{T-Bool-false})$$

$$\frac{}{\vdash \text{int\_expr}(\_) : \texttt{Int}} \quad (\text{T-Int})$$

$$\frac{}{\vdash \text{float\_expr}(\_) : \texttt{Double}} \quad (\text{T-Double})$$

$$\frac{\Gamma \vdash e_1 : \texttt{T} \text{ where } \texttt{T} \in \{\texttt{Int}, \texttt{Double}\}}{\Gamma \vdash -e_1 : \texttt{T}} \quad (\text{T-Neg})$$

$$\frac{\Gamma \vdash e_1 : \texttt{Bool}}{\Gamma \vdash \texttt{not}(e_1) : \texttt{Bool}} \quad (\text{T-Not})$$

$$\frac{\Gamma \vdash e_1 : \texttt{T}, n : \texttt{Int}}{\Gamma \vdash \texttt{Array::make}(n, e_1) : \texttt{Array}[\texttt{T}]} \quad (\text{T-Array-make})$$

§3.1.2. Arithmetics

$$\frac{\Gamma \vdash e_1 : \texttt{T}, e_2 : \texttt{T} \text{ where } \texttt{T} \in \{\texttt{Int}, \texttt{Double}\}}{\Gamma \vdash e_1 + e_2 : \texttt{T}} \quad (\text{T-Add})$$

$$\frac{\Gamma \vdash e_1 : \texttt{T}, e_2 : \texttt{T} \text{ where } \texttt{T} \in \{\texttt{Int}, \texttt{Double}\}}{\Gamma \vdash e_1 - e_2 : \texttt{T}} \quad (\text{T-Sub})$$

$$\frac{\Gamma \vdash e_1 : \texttt{T}, e_2 : \texttt{T} \text{ where } \texttt{T} \in \{\texttt{Int}, \texttt{Double}\}}{\Gamma \vdash e_1 * e_2 : \texttt{T}} \quad (\text{T-Mul})$$

$$\frac{\Gamma \vdash e_1 : \texttt{T}, e_2 : \texttt{T} \text{ where } \texttt{T} \in \{\texttt{Int}, \texttt{Double}\}}{\Gamma \vdash e_1 / e_2 : \texttt{T}} \quad (\text{T-Div})$$

§3.1.3. Comparisons

$$\frac{\Gamma \vdash e_1 : \texttt{T}, e_2 : \texttt{T} \text{ where } \texttt{T} \in \{\texttt{Int}, \texttt{Double}, \texttt{Bool}\}}{\Gamma \vdash e_1 \texttt{ <= } e_2 : \texttt{Bool}} \quad (\text{T-Leq})$$

$$\frac{\Gamma \vdash e_1 : \texttt{T}, e_2 : \texttt{T} \text{ where } \texttt{T} \in \{\texttt{Int}, \texttt{Double}, \texttt{Bool}\}}{\Gamma \vdash e_1 \texttt{ == } e_2 : \texttt{Bool}} \quad (\text{T-Eq})$$

§3.1.4. Get and apply

$$\frac{\Gamma \vdash e_1 : \texttt{Array}[\texttt{T}], e_2 : \texttt{Int}}{\Gamma \vdash e_1[e_2] : \texttt{T}} \quad (\text{T-Array-get})$$

$$\frac{\Gamma \vdash f : (\texttt{T}_1, \ldots \texttt{T}_n) \to \texttt{T}, e_1 : \texttt{T}_1, \ldots, e_n : \texttt{T}_n}{\Gamma \vdash f(e_1, \ldots, e_n) : \texttt{T}} \quad (\text{T-Apply})$$

§3.1.5. If

$$\frac{\Gamma \vdash e_1 : \texttt{Bool}, e_2 : \texttt{T}, e_3 : \texttt{T}}{\Gamma \vdash \texttt{if}(e_1, e_2, e_3) : \texttt{T}} \quad (\textsc{T-If})$$

§3.1.6. Statements

$$\frac{\Gamma \vdash e_1 : \texttt{Array}[\texttt{T}], e_2 : \texttt{Int}, e_3 : \texttt{T}, s_1 : \texttt{T}_s}{\Gamma \vdash e_1[e_2] = e_3; s_1 : \texttt{T}_s} \quad (\textsc{T-Array-put})$$

$$\frac{\Gamma \vdash e_1 : \texttt{T} \qquad \Gamma, v : \texttt{T} \vdash s_1 : \texttt{T}_s}{\Gamma \vdash \texttt{let}\ v = e_1; s_1 : \texttt{T}_s} \quad (\textsc{T-Let-var-untyped})$$

$$\frac{\Gamma \vdash e : \texttt{T} \qquad \Gamma, v : \texttt{T} \vdash s_1 : \texttt{T}_s}{\Gamma \vdash \texttt{let}\ v : \texttt{T} = e; s_1 : \texttt{T}_s} \quad (\textsc{T-Let-var-typed})$$

$$\frac{\Gamma \vdash e_1 : \texttt{Tuple}[\texttt{T}_1, ..., \texttt{T}_n] \qquad \Gamma, v_1 : \texttt{T}_1, ..., v_n : \texttt{T}_n, \vdash s_1 : \texttt{T}_s}{\Gamma \vdash \texttt{let}(v_1, ..., v_n) = e_1; s_1 : \texttt{T}_s} \quad (\textsc{T-Let-tuple-untyped})$$

$$\frac{\Gamma \vdash e_1 : \texttt{Tuple}[\texttt{T}_1, ..., \texttt{T}_n] \qquad \Gamma, v_1 : \texttt{T}_1, ..., v_n : \texttt{T}_n, \vdash s_1 : \texttt{T}_s}{\Gamma \vdash \texttt{let}(v_1, ..., v_n) : (\texttt{T}_1, ..., \texttt{T}_n) = e_1; s_1 : \texttt{T}_s} \quad (\textsc{T-Let-tuple-typed})$$

§3.1.7. Functions

Functions' parameters may not have types annotated, so type inference is needed. Fresh type variables are marked as X, and you can refer to Chapter 22 in TAPL on how to infer types.

$$\frac{\Gamma, ..., p_i : \texttt{T}_i, ... \vdash s_1 : \texttt{T}_r}{\Gamma \vdash \texttt{fn}\ f(..., p_i : \texttt{T}_i, ...)\{s_1\} : (..., \texttt{T}_i, ...) \to \texttt{T}_r} \quad (\textsc{T-Fn-param-typed})$$

$$\frac{\Gamma, ..., p_i : \texttt{X}, ... \vdash s_1 : \texttt{T}_r}{\Gamma \vdash \texttt{fn}\ f(..., p_i, ...)\{s_1\} : (..., \texttt{X}, ...) \to \texttt{T}_r} \quad (\textsc{T-Fn-param-untyped})$$

$$\frac{\Gamma \vdash s_1 : \texttt{Unit}}{\Gamma \vdash \texttt{fn main}\ \{s_1\} : () \to \texttt{Unit}} \quad (\textsc{T-Fn-main})$$

$$\frac{\Gamma \vdash s_1 : \texttt{Unit}}{\Gamma \vdash \texttt{fn init}\ \{s_1\} : () \to \texttt{Unit}} \quad (\textsc{T-Fn-init})$$